MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1024

April 1988

Model-Based Robot Learning

Christopher G. Atkeson, Eric W. Aboaf,
Joseph McIntyre and David J. Reinkensmeyer

Abstract: Models play an important role in learning from pr ice. Models of a controlled system can be used as learning operators to refine commands on the basis of performance errors. The examples used to demonstrate this include positioning a limb at a visual target, and following a defined trajectory. Better models lead to faster correction of command errors, requiring less practice to attain a given level of performance. The benefits of accurate modeling are improved performance in all aspects of control, while the risks of inadequate modeling are poor learning performance, or even degradation of performance with practice.

This report describes research done at the Artificial Intelligence Laboratory and Whitaker College, Department of Brain and Cognitive Sciences, of the Massachusetts Institute of Technology. Support for the Artificial Intelligence Laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124, and the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685. Support was provided in the Whitaker College by grants AM26710 and NS09343 from the NIH. Support for Chris Atkeson was provided by a Whitaker Health Sciences Fund MIT Faculty Research Grant. Support for Eric Aboaf was provided by an Office of Naval Research Graduate Student Fellowship. Support for Joe McIntyre was provided by the NIH training grant NIH-GM07484 and a Whitaker Health Sciences Fund Doctoral Fellowship. This paper appeared in the Proceedings of the Fourth International Symposium on Robotics Research, 1988.

Model-Based Robot Learning

Christopher G. Atkeson, Eric W. Aboaf,
Joseph McIntyre and David J. Reinkensmeyer

1 Introduction

An important component of human motor skill is the ability to improve performance by practicing a task. Commands are refined on the basis of performance errors. It is often suggested that such learning reduces the need for an accurate internal model, a model of the mechanical plant in the control system (see Arimoto, 1984b; Wang and Horowitz, 1985; and Harokopos, 1986 for examples). This is not the case. Internal models play an important role in generating command corrections from performance errors. As an internal model is made more accurate, learning efficiency is improved, as is initial performance.

This paper will show, in a series of examples, how internal models can be used as learning operators. The examples are 1) positioning a limb at a visual target, 2) throwing a ball at a target, and 3) following a defined trajectory. The essence of the model-based learning algorithms used to improve performance on these tasks is that internal models are used to transform performance errors into command corrections.

The type of learning described in this paper – refining commands on the basis of practice – complements many other types of adaptive processes. Feedback controller designs can be improved by adaptive control algorithms. Internal models can be incrementally improved using system identification techniques. Trajectories can be optimized for particular tasks. Robot plans and programs can be debugged as errors are discovered during execution. This paper focuses on improving execution of a given task plan by refining the commands given to the robot.

Model-Based Learning Algorithm Structure

The model-based learning algorithms described here all have the same form. Commands are refined on the basis of performance errors. A command is applied to the controlled system (Figure 1A). Performance errors may result from errors in the command. A model of the inverse of the controlled system is used to estimate the errors in the command based on the measured performance or output errors (Figure 1B). If the inverse model of the controlled system is perfect, the command errors would be correctly estimated and completely eliminated after one attempt at performing the task. (Of course, if a perfect model of the controlled system is available then the initial command would also have been perfect). Perfect knowledge of the controlled system is not usually available, and the model of the inverse of the controlled system will be incorrect. Due to the modeling errors, the command correction will be incomplete, and learning will be an iterative process of refining the command.

There are three steps to the learning algorithms: command initialization, execution, and modification. The initial command is generated by applying the inverse model of the controlled system to the desired performance. During execution, a command is applied to the system and the actual performance is monitored. The command correction is calculated by applying the inverse model to the performance errors. The refined command is now executed. The cycle of command execution and modification is repeated until desired performance is achieved.

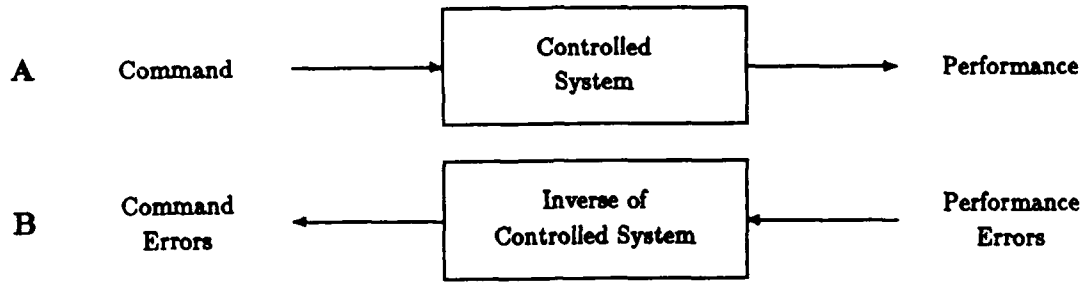

Figure 1: The inverse of the controlled system is used to estimate command errors from performance errors.

2 A Kinematic Example

The task of positioning the limb at a visual target will be used to provide a specific example of how model-based learning works. A robot arm and a target are viewed by a vision system (Figure 2). The robot arm servos to a commanded set of joint angles, θ, and the vision system measures the tip position, $\mathbf{x}$, in vision system coordinates. The controlled system in this case transforms commanded joint angles into a measured tip position (Figure 1A):

$$\mathbf{x} = L(\theta) \tag{1}$$

The forward kinematics, $L()$, is in general a nonlinear transformation. For the purposes of this example we will assume there are no singularities or redundancies to resolve in the field of view of the vision system. For each desired tip position there is one and only one appropriate set of joint angles.

A *model* of the inverse kinematics is used to transform the desired tip position, $\mathbf{x}_d$, into an initial joint angle command, θ^0, in the command initialization stage:

$$\theta^0 = \hat{L}^{-1}(\mathbf{x}_d) \tag{2}$$

A caret ($\hat{}$) is used to indicate a model or an estimate of a quantity. The initial joint angle command is applied in the first execution stage, and the corresponding tip position is measured:

$$\mathbf{x}^0 = L(\theta^0) \tag{3}$$

The true system, $L()$, and its inverse are unknown, and only imperfect models are available. Due to modeling errors, the actual tip position, $\mathbf{x}^0$, will not match the desired tip position, $\mathbf{x}_d$.

At this point we must decide how to transform the measured tip position error into a correction to the set of commanded joint angles. Performance errors must be mapped into command corrections. The same model of the inverse kinematics that was used to generate the initial command, $\hat{L}^{-1}()$, will be used to estimate the command error (Figure 1B).

The command error, $\delta\theta$, is the difference between the currently commanded joint angles, θ^0, and the (unknown) correct set of joint angles, which will be indicated as θ^*. The command error can be computed in terms of the actual and desired performances using the true system inverse:

$$\delta\theta^0 = \theta^0 - \theta^* = L^{-1}(\mathbf{x}^0) - L^{-1}(\mathbf{x}_d) \tag{4}$$

As we do not have perfect knowledge of the true system inverse, we must use a model of the system inverse to estimate the command error:

$$\widehat{\delta\theta}^0 = \hat{L}^{-1}(\mathbf{x}^0) - \hat{L}^{-1}(\mathbf{x}_d) \tag{5}$$

The command is updated by simply subtracting the estimate of the command error from the previous command:

$$\theta^1 = \theta^0 - \widehat{\delta\theta}^0 \tag{6}$$

If the model of the system inverse was perfect the command error would be estimated correctly and completely eliminated on the next attempt. However, a model is rarely perfect, so command correction must be an iterative process of estimating a command error using an imperfect model, removing the estimated command error, applying the refined command, and using the resulting performance error and the model to estimate remaining errors in the command. Equations (3), (5), and (6) can be indexed with i to indicate that they are applied on each practice attempt, reflecting the iterative nature of the algorithm:

1. Command initialization:

$$\theta^0 = \hat{L}^{-1}(\mathbf{x}_d) \tag{7}$$

2. Command execution:

$$\mathbf{x}^i = L(\theta^i) \tag{8}$$

3. Command error estimation:

$$\widehat{\delta\theta}^i = \hat{L}^{-1}(\mathbf{x}^i) - \hat{L}^{-1}(\mathbf{x}_d) \tag{9}$$

4. Command modification:

$$\theta^{i+1} = \theta^i - \widehat{\delta\theta}^i \tag{10}$$

Steps 2, 3, and 4 are repeated until satisfactory performance is achieved.

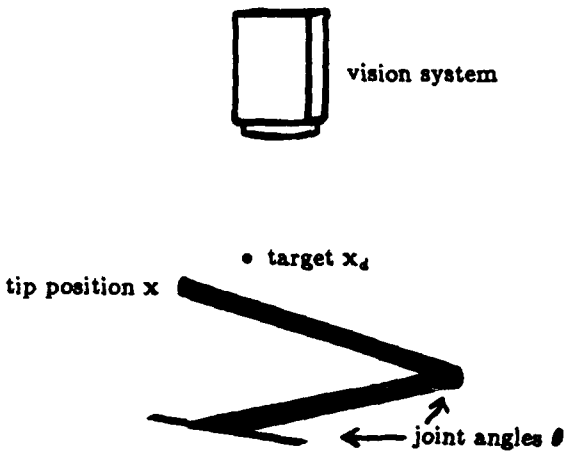

Figure 2: A robot arm and a target are viewed by a vision system.

Convergence

The quality of the inverse model used as the learning operator determines how fast model-based learning converges. Fixed point theory can be used to analyze the general nonlinear case (Wang 1984, Wang and Horowitz 1985). A learning algorithm can be viewed as a mapping of commands on the ith attempt to commands on the next attempt:

$$\theta^{i+1} = F(\theta^i) \tag{11}$$

The previously described algorithm can be put into this form by substituting equation (8) into (9) and (9) into (10). The model-based learning algorithm modifies the ith command by adding a correction based on the performance error transformed by the inverse model:

$$\theta^{i+1} = \theta^i - \left(\hat{L}^{-1}(L(\theta^i)) - \hat{L}^{-1}(x_d)\right) \tag{12}$$

Note that when the desired performance, x_d, is achieved using the correct command, θ^*, then $L(\theta^*) = x_d$ and equation (12) reduces to the fixed point $\theta^{i+1} = \theta^i = \theta^*$.

We can ask whether this fixed point is stable by analyzing a linearization of equation (12) at the point $(\theta, x) = (\theta^*, x_d)$. For a small perturbation $\delta\theta$ from the fixed point,

$$L(\theta^* + \delta\theta) = x_d + J(\theta^*)\delta\theta \tag{13}$$

where J is the Jacobean matrix of derivatives of $L()$. Similarly, for a small perturbation δx from the fixed point,

$$\hat{L}^{-1}(x_d + \delta x) = \hat{L}^{-1}(x_d) + \hat{J}^{-1}(x_d)\delta x \tag{14}$$

where $\hat{J}^{-1}$ is the Jacobean matrix for the inverse model $\hat{L}^{-1}()$. If on the ith trial the command is perturbed from θ^* by $\delta\theta^i$ so that $\theta^i = \theta^* + \delta\theta^i$, the error in the next command, $\delta\theta^{i+1} = \theta^{i+1} - \theta^*$, can be computed by substituting equations (13) and (14) into equation (12):

$$\delta\theta^{i+1} = (1 - \hat{J}^{-1}(x_d)J(\theta^*))\delta\theta^i \tag{15}$$

If $\hat{J}^{-1}$ is a correct inverse of J the command error will be completely corrected after one attempt, in the linear case. The command error $\delta\theta$ will decrease when all of the eigenvalues of the matrix $(1 - \hat{J}^{-1}J)$ are less than one in absolute value, with the rate of decrease determined by the magnitude of the eigenvalues. If the magnitude of any eigenvalue is greater than one, the learning process will be unstable and performance degraded rather than improved by learning. The magnitude of the eigenvalues of $(1 - \hat{J}^{-1}J)$ depend on how accurately $\hat{J}^{-1}$ inverts J, and thus the convergence rate of the learning algorithm depends on how closely the learning operator inverts the controlled system.

Input vs. output disturbance estimation

Although our performance errors are due to errors in modeling the controlled system, the model-based learning algorithm was derived by assuming that an unknown error was added to the command. In the kinematic tip positioning example a constant command disturbance would correspond to constant joint angle offsets added to the commanded joint angles. The learning algorithm just described can be viewed as an iterative procedure to estimate a command disturbance.

An alternative version of the model-based learning algorithm is suggested by assuming that the major source of errors are output (performance) disturbances rather than input (command) disturbances. In the kinematic example just presented, the camera measuring tip position could have an unknown offset, Δ. This offset could initially be assumed to be zero, and after each positioning attempt an estimate of the offset could be refined by subtracting the tip position error:

$$\Delta^i = \Delta^{i-1} - (x^{i-1} - x_d) \tag{16}$$

The estimated output offset would be added to the desired tip position when the next joint angle command was computed:

$$\theta^i = \hat{L}^{-1}(x_d + \Delta^i) \tag{17}$$

Equations (16) and (17) replace equations (9) and (10) in the input disturbance version of the model-based learning algorithm to form the output disturbance version.

Representing possible modeling errors as either input or output errors is a modeling decision that depends on the assumed source of the modeling errors. In the output disturbance version of the model-based learning algorithm, as in the input disturbance version, the performance error is mapped through an inverse model of the controlled system to calculate a command correction. The output disturbance model-based learning algorithm has similar convergence properties as the input disturbance algorithm.

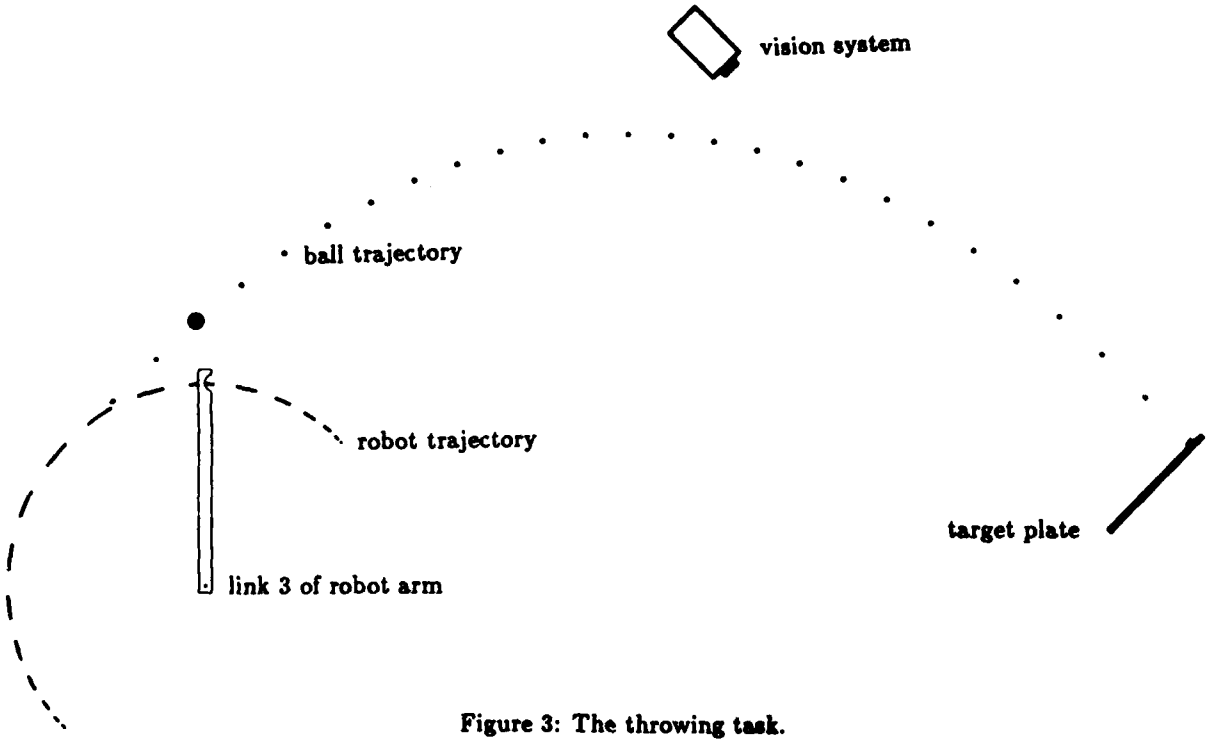

Figure 3: The throwing task.

3 Learning to Throw

Model-based learning can be used to improve performance on a complete task, in addition to improving positioning. As an example of task level learning, a robot arm was programmed to throw balls at a target. The robot throwing accuracy improved with practice.

Figure 3 illustrates the apparatus used in the throwing experiments. The target was at the center of a large metal plate, which was placed approximately 5 meters from the base of the robot. For this throwing task only the height of the ball when it hit the target plate was monitored and improved by a learning algorithm.

The last link of a three joint direct drive arm was used as a catapult to throw a ball. The robot was positioned so that the last link of the arm rotated in a vertical plane. The last joint was servoed to a fifth order polynomial trajectory that began at rest at 225° and ended at rest at 45°. A $4cm$ diameter rubber ball was placed onto a $3.5cm$ diameter hole at the end of the last link. The ball left the hole as the robot arm decelerated during the throw. No release mechanism was used. The release position of the ball was assumed to be when the last link was at 135°. The distance the ball was thrown was controlled by changing the duration of the throwing movement, which changed the release velocity. A shorter duration and therefore faster movement threw the ball higher and further, and a longer duration movement threw the ball lower and closer.

A video camera was used to record where the ball hit the target plate. The impact of the ball was sensed by a force sensor on which the target plate was mounted. This signal was used to choose video frames to be stored for later analysis. After the throw, the location of the ball on the target plate was manually measured from the appropriate video frame.

The initial release velocity command was calculated by measuring the distance to the target and using a simple ballistics model, incorporating only gravity, to predict the required flight trajectory given the assumed release position and initial direction of ball flight. The corresponding trajectory duration was computed and the calculated trajectory executed. On the first throw the ball hit the target plate $28cm$ above the target. The model-based learning algorithm based on estimating an output offset (equations (16) and (17)) was used to improve performance on the throwing task. This output offset learning algorithm corresponds to our intuition that we should aim lower if we are hitting too high, and vice versa. The role of the internal model is to calculate how much the aim should be changed. The ballistics model used to generate initial performance was also used to calculate the appropriate release velocity as the aim was offset by the estimated disturbance amount. The open squares in Figure 4 show the throwing performance during model-based learning. In this particular experiment the ball hit the target on the eighth throw.

The open triangles in Figure 4 indicate the performance of a model-based learning algorithm that improves the model as well as refining the command. This algorithm will be discussed in a later paper.

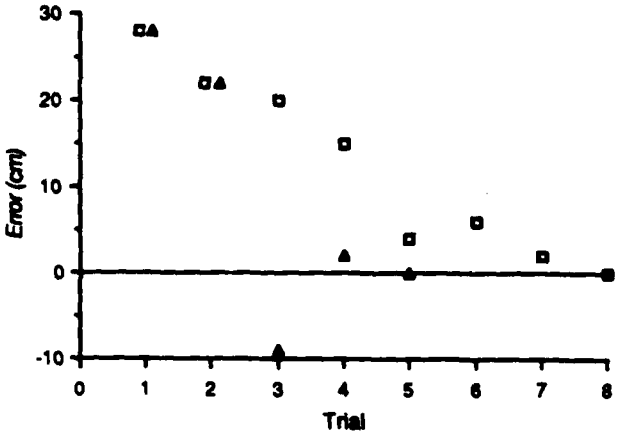

Figure 4: Performance of the model-based learning algorithm on a throwing task.

4 Trajectory Learning

Trajectory execution of a robot can be improved using a model-based learning algorithm (Atkeson and McIntyre 1986a, 1986b). A model of the robot inverse dynamics is used as the learning operator that transforms trajectory following errors into feedforward command corrections. This form of learning is useful for refining repetitive motions, and can also be used to refine groups of similar motions. Model-based trajectory learning was implemented on the MIT Serial Link Direct Drive Arm and greatly reduced trajectory following errors in a small number of practice movements.

The robot model used as the learning operator in the trajectory learning experiments was identified from movements of the MIT Serial Link Direct Drive Robot Arm (Atkeson, An, and Hollerbach 1986). The dynamics of this direct drive robot arm are dominated by rigid body dynamics, so a Newton-Euler model structure was used. The Newton-Euler rigid body dynamics equations for a robot can be written as

$$\tau = \hat{R}^{-1}(\theta, \dot{\theta}, \ddot{\theta}) = \mathbf{I}(\theta) \cdot \ddot{\theta} + \dot{\theta} \cdot \mathbf{C}(\theta) \cdot \dot{\theta} + \mathbf{g}(\theta) \quad (18)$$

where $\theta(t)$ is the desired trajectory of the joint angles, $\tau(t)$ is the vector of required torques to achieve the desired trajectory, $\mathbf{I}(\theta)$ is the inertia matrix of the arm, $\mathbf{C}(\theta)$ is the Coriolis and centripetal force tensor, and $\mathbf{g}(\theta)$ is the gravitational force vector (Hollerbach, 1984). For other types of robots it is argued that additional sources of dynamics are important (Goor, 1985; Good, Sweet, and Strobel, 1985). In these cases we can still model the robot dynamics and invert the model.

As before, there are several stages of the algorithm. The initial feedforward command is generated by applying the model of the robot inverse dynamics to the desired trajectory (as in equation (7)):

$$\tau_{ff}^0(t) = \hat{R}^{-1}(\theta_d(t), \dot{\theta}_d(t), \ddot{\theta}_d(t)) \quad (19)$$

During command execution the applied command is the sum of the feedforward command, τ_{ff}, and the output of the feedback controller, τ_{fb}:

$$\tau^i(t) = \tau_{ff}^i(t) + \tau_{fb}^i(t) \quad (20)$$

The total applied command, τ, is used as the basis for the next feedforward command. As described in the previous sections, the command error is estimated using the model of the robot inverse dynamics (as in equation (9)):

$$\widehat{\delta\tau}^i(t) = \hat{R}^{-1}(\theta^i(t), \dot{\theta}^i(t), \ddot{\theta}^i(t)) - \hat{R}^{-1}(\theta_d(t), \dot{\theta}_d(t), \ddot{\theta}_d(t)) \quad (21)$$

and the next feedforward command is the modified total command (as in equation (10)):

$$\tau_{ff}^{i+1}(t) = \tau^i(t) - \widehat{\delta\tau}^i(t) \quad (22)$$

Other Approaches to Trajectory Learning

Recent work in a number of laboratories has focused on how to refine feedforward commands for repetitive movements on the basis of previous movement errors. Work on repeated trajectory learning includes (Arimoto et al 1984, 1985; Casalino & Gambardella 1986; Craig 1984; Furuta & Yamakita 1986; Hara et al 1985; Harokopos 1986; Mita & Kato 1985; Morita 1986; Togai & Yamano 1986; Uchiyama 1978; Wang 1984; Wang & Horowitz 1985). These papers discuss only linear learning operators and emphasize the stability of the proposed algorithms. There has been little work emphasizing performance, i.e. the convergence rate of the algorithm. Simulations of several of these algorithms have revealed very slow convergence and large sensitivity to disturbances and sensor and actuator noise (C. G. Atkeson, unpublished results).

An Implementation of the Trajectory Learning Algorithm

The model-based trajectory learning algorithm has been implemented on the MIT Serial Link Direct Drive Arm (Atkeson and McIntyre 1986a, 1986b). This three joint arm is described in (Atkeson, An, and Hollerbach 1986). To explore the effectiveness of the model-based trajectory learning algorithm we will present results on learning a particular trajectory.

The Test Trajectory: All three joints of the Direct Drive Arm were commanded to follow a fifth order polynomial trajectory with zero initial and final velocities and accelerations and a 1.5 second duration. Figure 5 shows the shape of the trajectory for each joint, and Table 1 gives the initial and final joint positions, the peak joint velocities, and the peak joint accelerations.

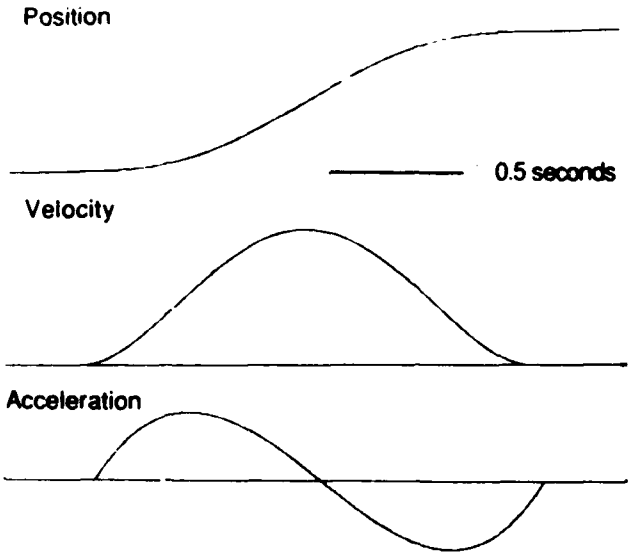

Figure 5: The test trajectory.

Joint	Initial Position *radians*	Final Position *radians*	Peak Velocity *radians/s*	Peak Acceleration *radians/s²*
1	0.5	4.5	5.0	±10.3
2	5.0	1.0	-5.0	±10.3
3	4.0	-0.5	-5.6	±11.5

Table 1: Test trajectory parameters.

The Feedback Controller: An independent digital feedback controller was implemented for each joint and was not modified during learning.

Initialization Of The Feedforward Command: The initial feedforward torques were generated from a rigid body dynamics model. The model and the estimation of its parameters are described in (Atkeson, An, and Hollerbach, 1986). The calculated feedforward torques are shown in Figure 6A.

Initial Trajectory Performance: As an index of trajectory following performance the velocity errors (the difference between the actual joint velocity and the desired joint velocity) for the first movement are shown in Figure 7A. We have plotted the raw velocity error data to give an idea of the relative size of the trajectory errors and sensor noise.

Calculating Acceleration and Filtering: In order to use the rigid body inverse dynamics model to compute joint torques it was necessary to compute the joint accelerations. Joint positions and velocities were measured directly. A digital differentiating filter combined with an 8Hz low pass filter was applied to the velocity data to estimate accelerations.

To reject noise and non-repeatable disturbances and to compensate for high frequency unmodelled dynamics it was necessary to filter the trajectory errors and controller output. In this implementation we applied low pass digital filters with an 8Hz cutoff to the data used in the learning process. We filtered the references used by the learning operator with the same filter used on the data. It was also necessary to correct for inconsistencies between the velocity sensors and the position measurements, which was done by adjusting the position reference to the feedback controller until the integrated velocity error matched the position error.

Final Trajectory Performance: The robot executed two additional training movements which are not shown, and its performance on the fourth attempt of the test trajectory was assessed. Figure 6B shows the modified feedforward commands used on the fourth movement, and should be compared with the predicted torques shown in Figure 6A. Figure 7B shows the velocity errors for the fourth movement, and should be compared with the initial movement velocity errors in Figure 7A. There has been a substantial reduction in trajectory following error after only three practice movements.

5 Issues For Further Research

Some of the questions that warrant further research include the effect of modeling errors and non-repeatable disturbances on convergence, and learning of non-repetitive tasks.

As discussed previously, the convergence of model-based learning algorithms depends on the quality of the model. Accurate models support efficient learning. Inaccurate models may cause learning algorithms to degrade performance rather than improve it.

Reducing or filtering the estimated command correction will make model-based learning more robust to modeling errors. Convergence will be slowed, however. Further research is required into the appropriate tradeoff between handling modeling errors and fast convergence. Filtering of the model-based command update also plays an important role in reducing the effect of non-repeatable disturbances.

If intermediate sensory signals are available, then breaking the control system into modules and having each module learn independently may improve learning performance. We plan to explore this issue in the throwing task. If measurements are available of when and where the ball is released, then independent models of the throwing motion and the ball flight characteristics can be made. These independent models can be used to choose an appropriate release velocity separately from refining the trajectory that attains that release velocity.

It is possible to modify models as well as commands during learning. In the examples presented in this paper the same models were used repeatedly even after it became clear during learning that the models had large errors. We have explored some methods of model refinement during practice. The open triangles of Figure 4 show the faster convergence of a model-based learning algorithm that improves the model as well as the command.

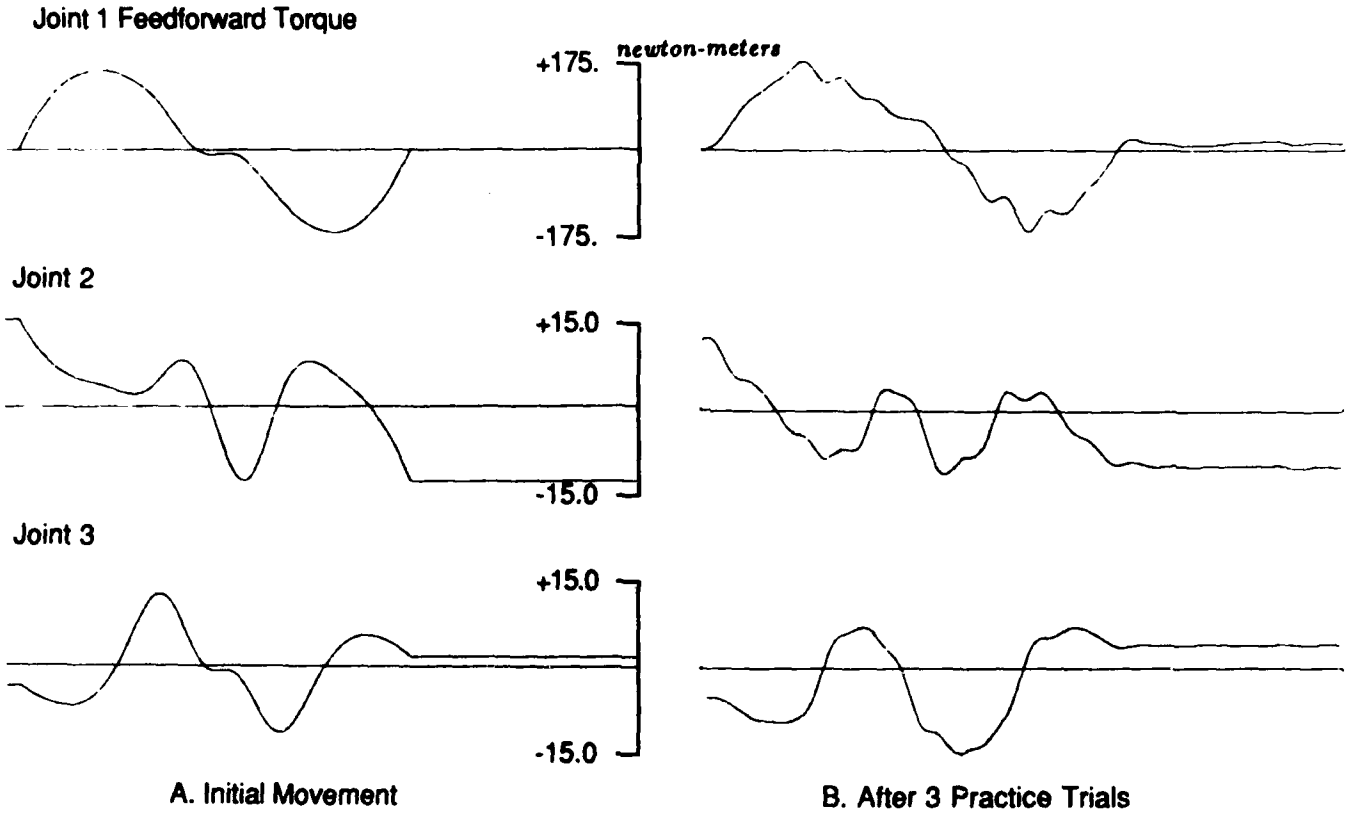

Figure 6: Feedforward Torques

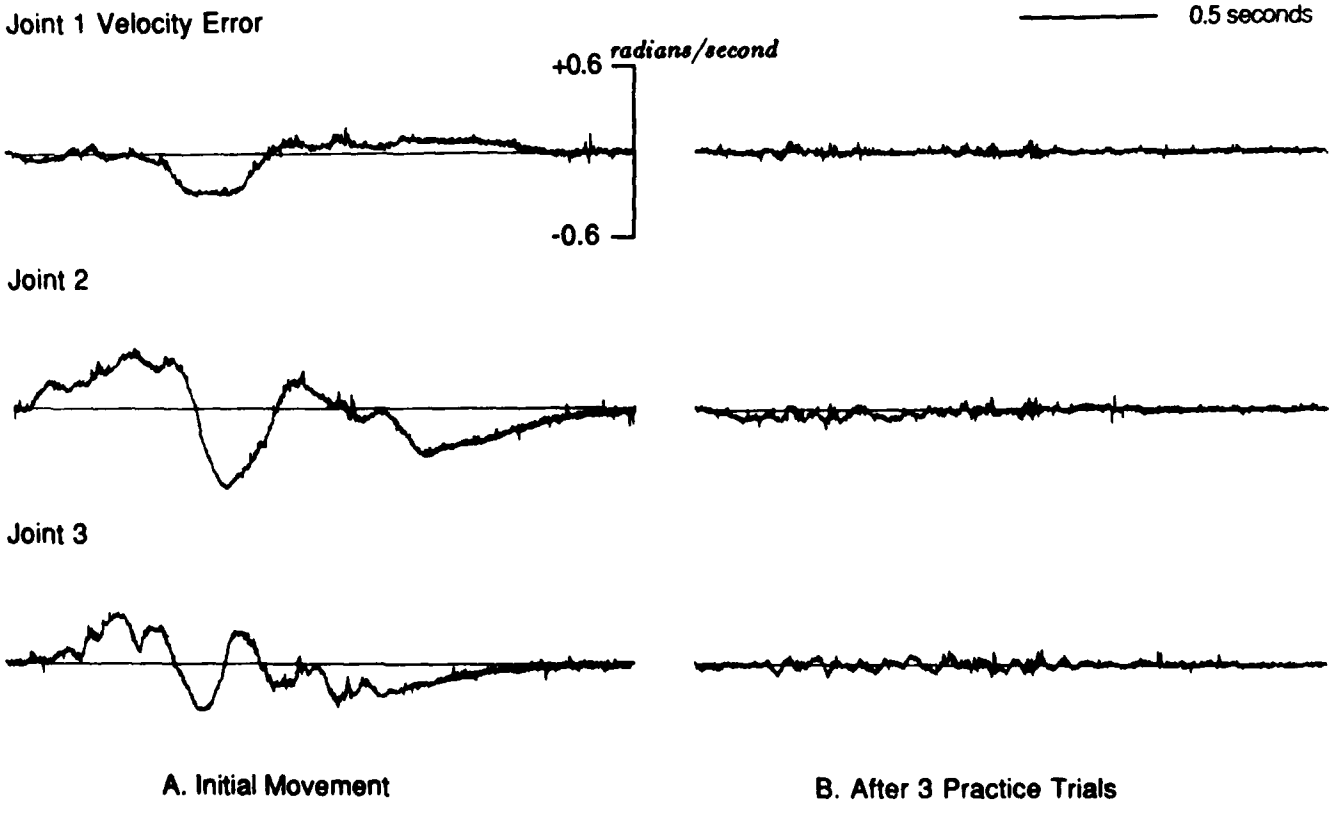

Figure 7: Velocity Errors

The model-based learning algorithms are ideally suited to refining repetitive commands for the same tasks. The learning algorithms can also be applied to refining commands for different tasks by assuming that similar command errors will be made on similar tasks. An estimate of the command error on one task will be useful for improving the command for other tasks that share features with the original task.

6 Conclusion

The main message of this paper is that models play an important role in learning from practice. Better models lead to faster correction of command errors. The incorporation of learning in a control system is not a license to do a poor modeling job of the controlled system. The benefits of accurate modeling are better performance in all aspects of control, while the risks of inadequate modeling are poor learning performance or even degradation of performance with practice.

The approach to robot learning presented here is based on explicit modeling of the robot and the task being performed. An inverse model of the task is used as the learning operator that processes the errors. Such model-based command refinement algorithms usefully complement other approaches to adaptive control.

Studying model-based learning algorithms serves two purposes: 1) to improve robot performance, and 2) to increase our understanding of the role of practice and internal models in human motor learning.

Acknowledgments

This article describes research done at the Whitaker College, Department of Brain and Cognitive Sciences, and the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support was provided in the Whitaker College by grants AM26710 and NS09343 from the NIH. Support for the A. I. Laboratory's research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124, and the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685. Support for CGA was provided by a Whitaker Health Sciences Fund MIT Faculty Research Grant. Support for EWA was provided by an Office of Naval Research Graduate Student Fellowship. Support for JM was provided by an NIH training grant NIH-GM07484 and a Whitaker Health Sciences Fund Doctoral Fellowship.

References

Arimoto, S., S. Kawamura, and F. Miyazaki, "Bettering Operation of Robots by Learning", *Journal of Robotic Systems* 1 (1984a) 123-140.

Arimoto, S., S. Kawamura, and F. Miyazaki, "Can Mechanical Robots Learn by Themselves?", Proc. of 2nd Inter. Symp. Robotics Research (Kyoto, Japan, August, 1984b).

Arimoto, S., S. Kawamura, F. Miyazaki, and S. Tamaki, "Learning Control Theory for Dynamical Systems", Proc. 24th Conf. on Decision and Control (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Atkeson, C. G., C. An, and J. M. Hollerbach, "Estimation of Inertial Parameters of Manipulator Loads and Links", *International Journal of Robotics Research* 5 (1986) 101-119.

Atkeson, C. G. and J. McIntyre, "Robot Trajectory Learning Through Practice", IEEE Conf. on Robotics and Automation (San Francisco, CA, April 7 - 10, 1986a).

Atkeson, C. G. and J. McIntyre, "Applications of Adaptive Feedforward Control in Robotics", Proc. 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing (Lund, Sweden, July 1-3, 1986b).

Casalino, G. and L. Gambardella, "Learning of Movements in Robotic Manipulators", Proc. 1986 IEEE International Conference on Robotics and Automation (San Francisco, CA, April 7-10, 1986a) pp. 572-578.

Craig, J. J., "Adaptive Control of Manipulators Through Repeated Trials", Proc. American Control Conference (San Diego, June 6-8, 1984) pp. 1566-1574.

Furuta, K. and M. Yamakita, "Iterative Generation of Optimal Input of a Manipulator", Proc. 1986 IEEE International Conference on Robotics and Automation (San Francisco, CA, April 7-10, 1986) pp. 579-584.

Good, M.C., Sweet, L.M., and Strobel, K.L., "Dynamic models for control system design of integrated robot and drive systems", *ASME J. Dynamic Systems, Meas., Control,* 107 (1985) 53-59.

Goor, R.M., "A new approach to robot control", Proc. American Control Conf (Boston, June 19-21, 1985) pp. 385-389.

Hara, S., T. Omata, and M. Nakano, "Synthesis of Repetitive Control Systems and its Application", Proc. 24th Conf. on Decision and Control (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Harokopos, E. G., "Optimal Learning Control of Mechanical Manipulators in Repetitive Motions", Proc. 1986 IEEE International Conference on Robotics and Automation (San Francisco, CA, April 7-10, 1986) pp. 396-401.

Hollerbach, J. M., "Dynamic Scaling of Manipulator Trajectories", *Journal of Dynamics Systems, Measurement, and Control* 106 (1984) 102-106.

Mita, T., and E. Kato, "Iterative Control and its Application to Motion Control of Robot Arm – A Direct Approach to Servo-Problems", Proc. 24th Conf. on Decision and Control (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Morita, A., "A Study of Learning Controllers For Robot Manipulators With Sparse Data", M.S. thesis, Mechanical Engineering, Massachusetts Institute of Technology (February 27, 1986).

Togai, M. and O. Yamano, "Learning Control and Its Optimality: Analysis and Is Application to Controlling Industrial Robots", Proc. 1986 IEEE International Conference on Robotics and Automation (San Francisco, CA, April 7-10, 1986) pp. 248-253.

Uchiyama, M., "Formation of High-Speed Motion Pattern of a Mechanical Arm by Trial", *Transactions of the Society of Instrument and Control Engineers (Japan)* 19 (1978) 706–712.

Wang, S. H., "Computed Reference Error Adjustment Technique (CREATE) for The Control of Robot Manipulators", 22nd Annual Allerton Conf. on Communication, Control, and Computing (October, 1984).

Wang, S. H., and I. Horowitz, "CREATE - A New Adaptive Technique", Proc. of the Nineteenth Annual Conference on Information Sciences and Systems (March, 1985).